To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

H8S/H8SX Families

Using the Graphics API to Implement Sliding Icons and Transparency

Introduction

Renesas provides a standard set of functions for creating and manipulating graphics and text on a TFT-LCD panel. These functions are referred to as the Graphics Application Programming Interface or GAPI.

This Application Note is one in a series of application examples which show how to implement interactive graphics on an LCD panel.

This Application Note can be used with any available Direct Drive LCD Demo PCB from Renesas.

Design manuals, software and schematics are also available from www.america.renesas.com/h8lcd.

This application note must be used in conjunction with the REU05b0112_H8SAP application note and code. Please download and install REU05b0112_H8SAP application note first and use the below instructions to add this source code to the project.

Target Device

H8S2378, H8S2456, H8SX1668R

and Direct Drive LCD Demo Board

Contents

1. Transparently moving a bitmap on the screen 2

2. Code 5

3. Touchscreen and Panel Coordinates 10

4. Installation and Source Code Structure 11

1. Transparently moving a bitmap on the screen

This sample code uses GAPI calls which will access the bitmap in memory and move it randomly about the screen.

This code also maps the bitmap to a touchable area based on its size, and specifies which function will be called in the event that the icon is touched.

This code moves the icon by drawing each image from the bitmap on a periodic basis.

Figure 1 shows the bitmap that we will be moving about the screen in the sample code:

Figure 1 BigR.BMP and ScreenBounce

1.1 Creating a transparent BMP compatible with GAPI

There are many tools that can create bitmap images compatible with GAPI. In this example we will use Adobe Photoshop® (raster editor) and Adobe Illustrator® (vector editor) to create a standard bitmap file image that we can use within GAPI. Often these tools are used in a corporate environment for high quality logos and images. In this case we wish to use the stylized “R” from the Renesas logo. The advantage of the .ai format is that it is vector based and can easily be scaled to any pixel size without loss of information (pixelization). To start we will scale our logo to the desired pixel size in the vector editor, and then copy this to the raster editor.

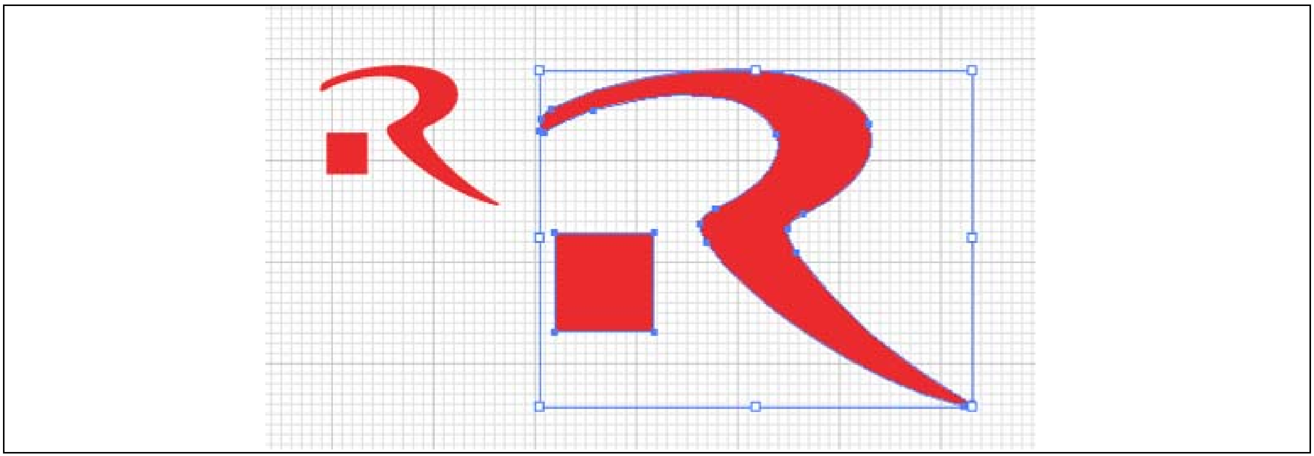


Figure 2 Vector Logo

When starting your image in the raster editor, start with an RGB color 8:8:8 image with transparent background. Paste the copied vector image into the raster editor. Make sure “anti-alias” is off for this process…this will provide solid edges that will blend against a random background. Next, we will convert the image mode to “indexed color” (image->mode->indexed color, merge layers OK. Depending on your particular image, the number of colors will vary, but ensure that “transparency” is checked in the creation of the color table (GAPI utilizes this transparency). You can now “save as” a standard bitmap (BMP) file. Given this file only has 4 colors in the color table, it can be saved as either a 4Bpp or 8Bpp indexed BMP format (GAPI supports both 4Bpp and 8Bpp indexed formats). Locating this file in the demo’s “Resources” directory will automatically cause it to be built into the “resources.bin” file with all other demo resources (refer to the REU05b0112 for more information on resources).

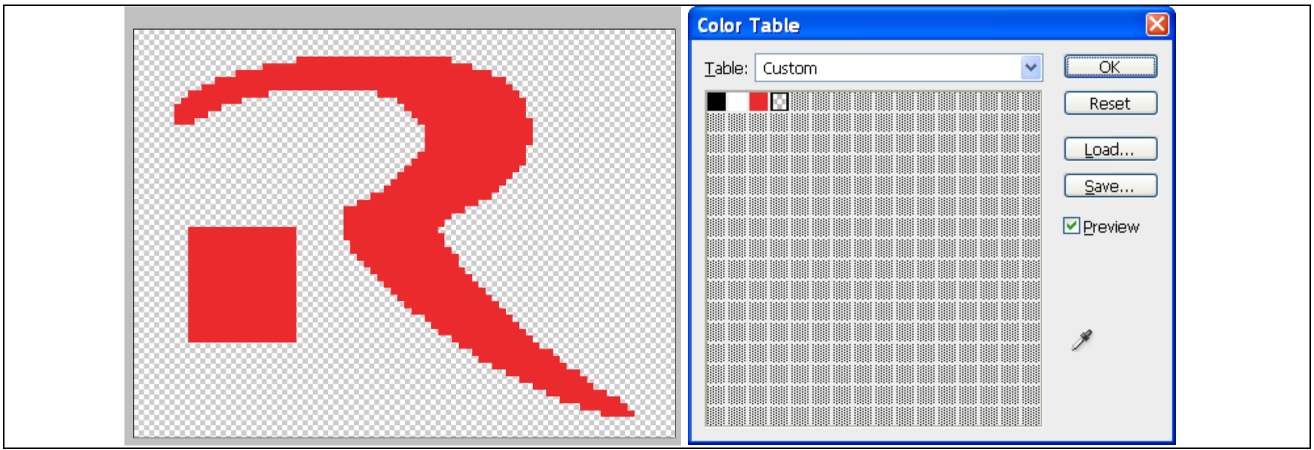
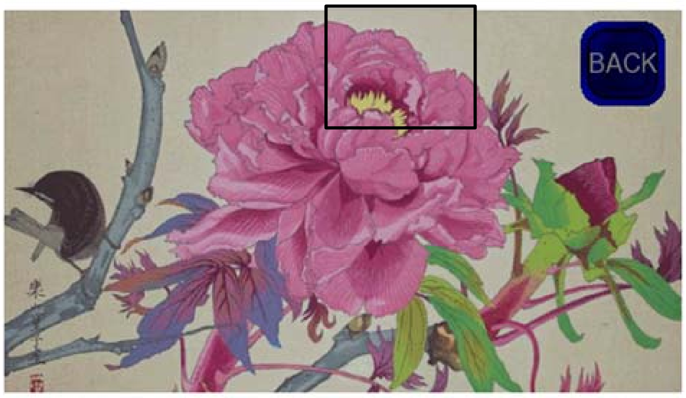


Figure 3 Raster Logo and color table

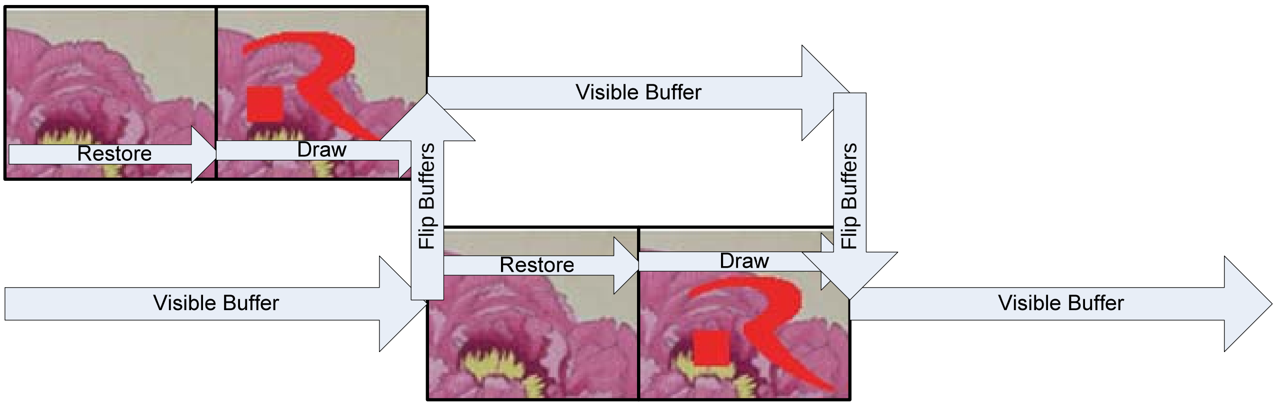
1.2 Animation by page flipping

There are many techniques that can be used to transparently move an image on a screen. In this demonstration we use a page flipping technique as illustrated below. When moving an image against a complex background, the prior location of the image must be restored before painting the image again in the new location. In this demo, we keep a reference frame (backFramebuf) that is used to restore corrupted areas. If we were to use a single buffer we would visually see the draw/restore/draw process and perceive "flicker" as sometimes we would see the restored area that was about to be drawn again. By using two buffers, we perform the restore/draw process on a non-visible buffer, then flip (make visible) this buffer only after completing the process. At this time, the previously visible buffer is available for the restore/draw process. To minimize the time consumed in the restore/draw process only the frame regions that are known to be affected are written to.



backFrameBuf (restore source)

Simple use of single frame buffer (flicker)



Page flipping with two frame buffers (no flicker)

2. Code

2.1 Setting Up the Icon

The Icons table describes which objects will be placed on the screen. The pBigR image will be placed at screen coordinates x = 0.0, y = VERTICAL_POS_PCT (0.396) (relative screen coordinates) when the "BasicConstructor" processes the "BounceLogo" callback function.

Any number of icons can be placed on the screen; the final entry in the table must be a NULL to terminate the list.

```
static const ICON_type Icons[]=
{
//*****************************************************************************
//BITMAP ADDR                            FUNCTION CALL            Y POSITION
//                      COLOR SCHEME                       X POSITION
//*****************************************************************************

  { &pBMP_ButtonS, T_SchemeBlue,   ButtonBack,     SX(0.850), SY(0.750) },
  { &pBMP_BigR,    T_SchemeNoColor,BounceLogo, SX(0.000),   SY(VERTICAL_POS_PCT)   },
  { NULL,              NULL,           NULL,           0,        0      },};

SCREEN_type ScreenHomeData=
{
  Constructor, Destructor, Icons
};
```

2.2 Screen Constructor/Destructor

Every screen is started by the framework calling the screen Constructor. In this sample code, the constructor first locates the "BigR" (BMP) file in the pResources structure and assigns this location to the "pBMP_BigR" handle. When the "BasicConstructor" is called, it will execute each Icon table callback function (these functions are responsible for placing their own graphic images). The BasicConstructor initializes the displayFrameBuf, but this code also utilizes the workFrameBuf for page flipping, so this code also initializes the workFrameBuf with a GAPI LCDBMPCopy call. Because we will be moving our BigR icon about the screen, we didn't place it into the backFrameBuf (as it would be copied to the displayFrameBuf during the restore process). So initial placement of the BigR in the displayFrameBuf is accomplished by a GAPI LCDBMPCopyTransparent call.

The "Destructor" function is called on exit from the screen by the framework. The "BasicDestructor" will release the screen task that we will be using.

```
static void Constructor(SCREEN_type const *pS)
{
  if(pBMP_Background == NULL)
    pBMP_Background = FileFind(pResources, "TestImage8bpp");

  if(pBMP_BigR == NULL)
    pBMP_BigR = FileFind(pResources, "BigR");

  //fill background
  (void)LCDBMPCopy(pBMP_Background, backFrameBuf, 0, 0);

  // Run default behavior
  BasicConstructor(pS);

  //initialize work frame buffer
  (void)LCDBMPCopy(backFrameBuf, workFrameBuf, 0, 0);

  //Display icon. BasicConstructor() would take care of this
  //normally, but since we are flipping buffers for animation it must
  //be drawn manually to the front buffer.
  (void) LCDBMPCopyTransparent(pBMP_BigR, displayFrameBuf, x_pos,   y_pos);
}

static void Destructor(SCREEN_type const *pS)
{
  // Run default behavior
  BasicDestructor(pS);
}
```

2.3 Callback function

The "BounceLogo" function will be called on every event that the system receives while this sample screen is active. It is the responsibility of this function to decide if it needs to process the event.

When "BounceLogo" is called by the "BasicConstructor", it will be passed a message ID of "MSG_DRAW". When this message is received we reset the "running" flag of the "BounceTask" (as all screen tasks are stopped on a screen change).

When "BounceLogo" is called because of touch events, it will check if the "BounceTask" is "running". If not, and the touch was within the boundary of the BigR, it will process "MSG_RELEASE" ID's. If so, we will start the "BounceTask" screen task responsible for periodically moving the image. If "BounceTask" is already "running", a "MSG_RELEASE" event anywhere on the screen will stop it.

```
void BounceLogo(ICON_type const *pS, EVENT_MSG const *pMsg)
{
  if(MSG_DRAW == pMsg->id)
  {
    /* reset state of icon */
    running = 0;
  }

  if(0 == running)
  {
    //press the logo to start the task
    //we want to check if the logo has been touched, but we do not want IconHandler()
    //to draw the logo for us.  That is being handled separately.
    if (((x_pos <= pMsg->param.coord[0]) &&
              (pMsg->param.coord[0] < (x_pos + BMP_Width(*pS->ppBmp)))) &&
        ((y_pos <= pMsg->param.coord[1]) &&
              (pMsg->param.coord[1] < (y_pos + BMP_Height(*pS->ppBmp)))))
    {
      if (MSG_RELEASE == pMsg->id)
      {
        //start screen task
        (void)ScreenTaskStart(BounceTask);
        running = 1;
      }
    }
  }
  else
  {
    //pressing anywhere else will stop the task
    if (MSG_RELEASE == pMsg->id)
    {
      //stop screen task
      (void)ScreenTaskStop(BounceTask);
      running = 0;
    }
  }
}
```

2.4 Moving the Bitmap

The “ BounceTask” function is called on a periodic basis (determined by the xDelay return value…50mS here). Because code in this thread will be accessing the external bus, we must “window” the usage with “ExMemoryAcquire” and “ExMemoryRelease” to prevent conflict with the direct driver.

First the code calculates a new x,y coordinate pair for the icon. This code just adds a fixed value (RATE) via “_move” variables, the sign of these variables is changed whenever the icon “bumps” against a screen edge to keep the “BigR” within the screen boundaries.

The code than restores the “clean” area from the backFrameBuf to the workFrameBuf by use of the GAPI LCDBMPCopySub call. This restoration only copies an area as big as the BigR bitmap. Then the BigR is painted at the new location with the GAPI LCDBMPCopyTransparent call.

Finally the code “flips” the display with the GAPI LCDBMP_WorkDisplay call (making workFrameBuf the displayFrameBuf and displayFrameBuf the workFrameBuf)

```
static TickType BounceTask()
{
  // Update image at 20Hz
  TickType xDelay = (TickType)(50/TICK_RATE_MS);

  /* Let system know we're accessing External Memory */
  ExMemoryAcquire(RLCD_GetTaskHandle());
  {
    //index of the currently active frame
    static sI16 index = 0;

    //this array hold previous x and y coordinates for each buffer
    static POINT_TYPE prev[2] = { {SX(0.000), SY(VERTICAL_POS_PCT)},
                                  {SX(0.000),   SY(VERTICAL_POS_PCT)} };

    //width and height of the image
    sI16 img_width = BMP_Width(pBMP_BigR);
    sI16 img_height = BMP_Height(pBMP_BigR);

    {
      /* move icon */
      static sI16 x_move=RATE, y_move=RATE;
      x_pos+=x_move;
      if (x_pos < 0)
        x_move = RATE;
      else if(x_pos > BMP_Width(backFrameBuf)-img_width)
        x_move = -RATE;

      y_pos+=y_move;
      if (y_pos < 0)
        y_move = RATE;
      else if(y_pos > BMP_Height(backFrameBuf)-img_height)
        y_move = -RATE;
    }

    //restore the area of this buffer corresponding to the image's previous location
    (void) LCDBMPCopySub( backFrameBuf, workFrameBuf,
                 prev[index].x, prev[index].y, prev[index].x, prev[index].y, img_width, img_height,
                 pBMP_Background->biColorTable, NO_TRANSPARENCY_COLOR);
    //write the image in its new position
    (void) LCDBMPCopyTransparent( pBMP_BigR, workFrameBuf, x_pos, y_pos);

    //store previous values
    prev[index].x = x_pos;
    prev[index].y = y_pos;
    //flip buffer
    index = LCDBMP_WorkDisplay();

  }
  ExMemoryRelease(RLCD_GetTaskHandle());

  return (xDelay);
}
```

3. Touchscreen and Panel Coordinates

By convention, the sample code uses relative screen coordinates. This is accomplished by use of the "SX" and "SY" macro expansions. These expansions convert normalized coordinates (0.00 to 1.00) to absolute screen coordinates. If desired, the SX/SY macros can not be used and absolute screen coordinates used.

For example SX(0.5), SY(0.5) on a QVGA (320x240) panel would expand to (160,120)

4. Installation and Source Code Structure

The code is contained within one source file called “ScreenBounce.c,” and the bitmap images used are contained in a bitmap file “BigR.bmp”. To install the sample code, double click on the installation executable “REUE05B0107.exe” to bring up the installer. (figure 2) Make sure you click on the browse button in the “Destination Folder” Panel and select your LCD Direct Drive demo project (REU05b0112_H8SAP installation) directory. Then click the next button to copy the new files into your project directory.

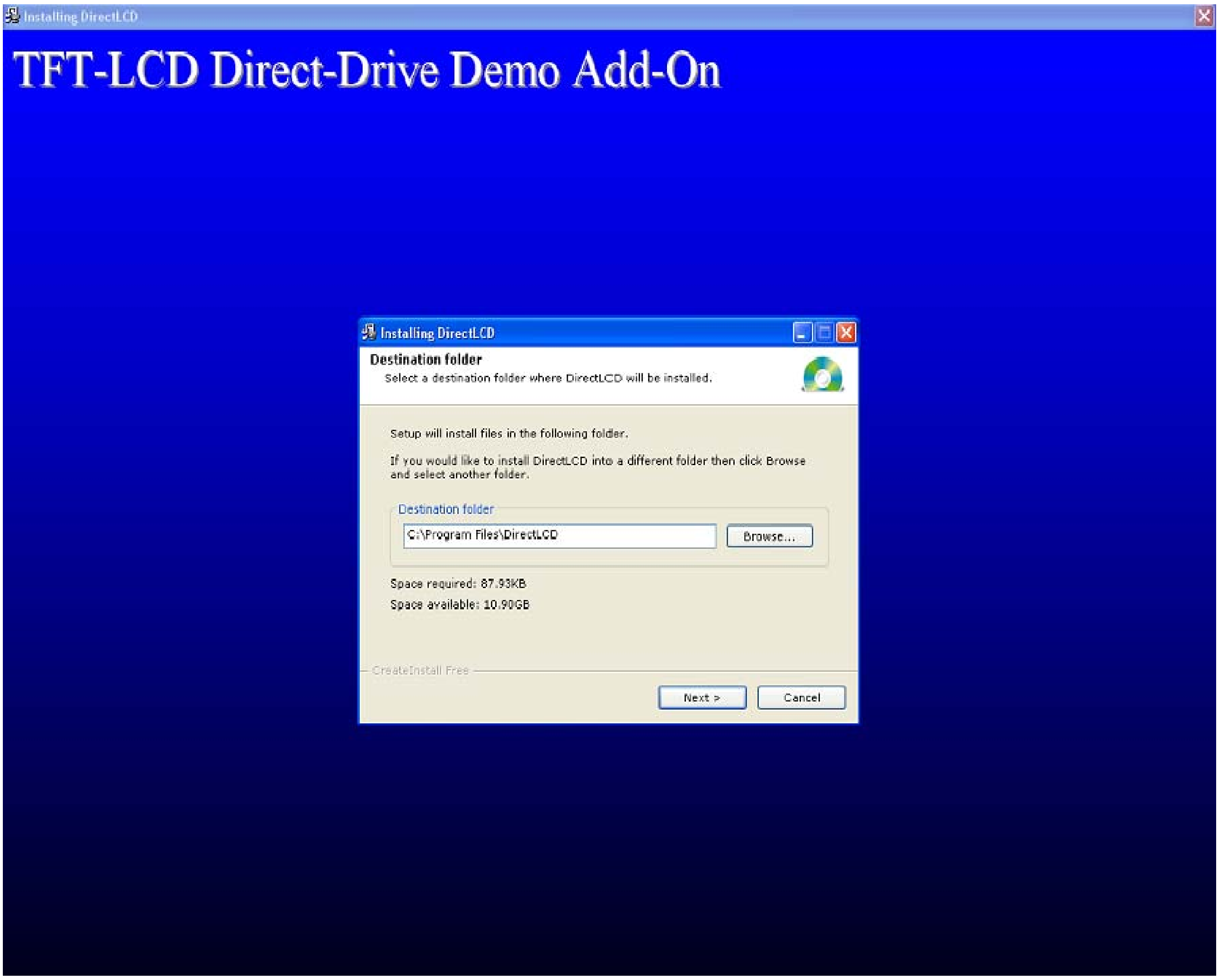


Figure 2: Installation

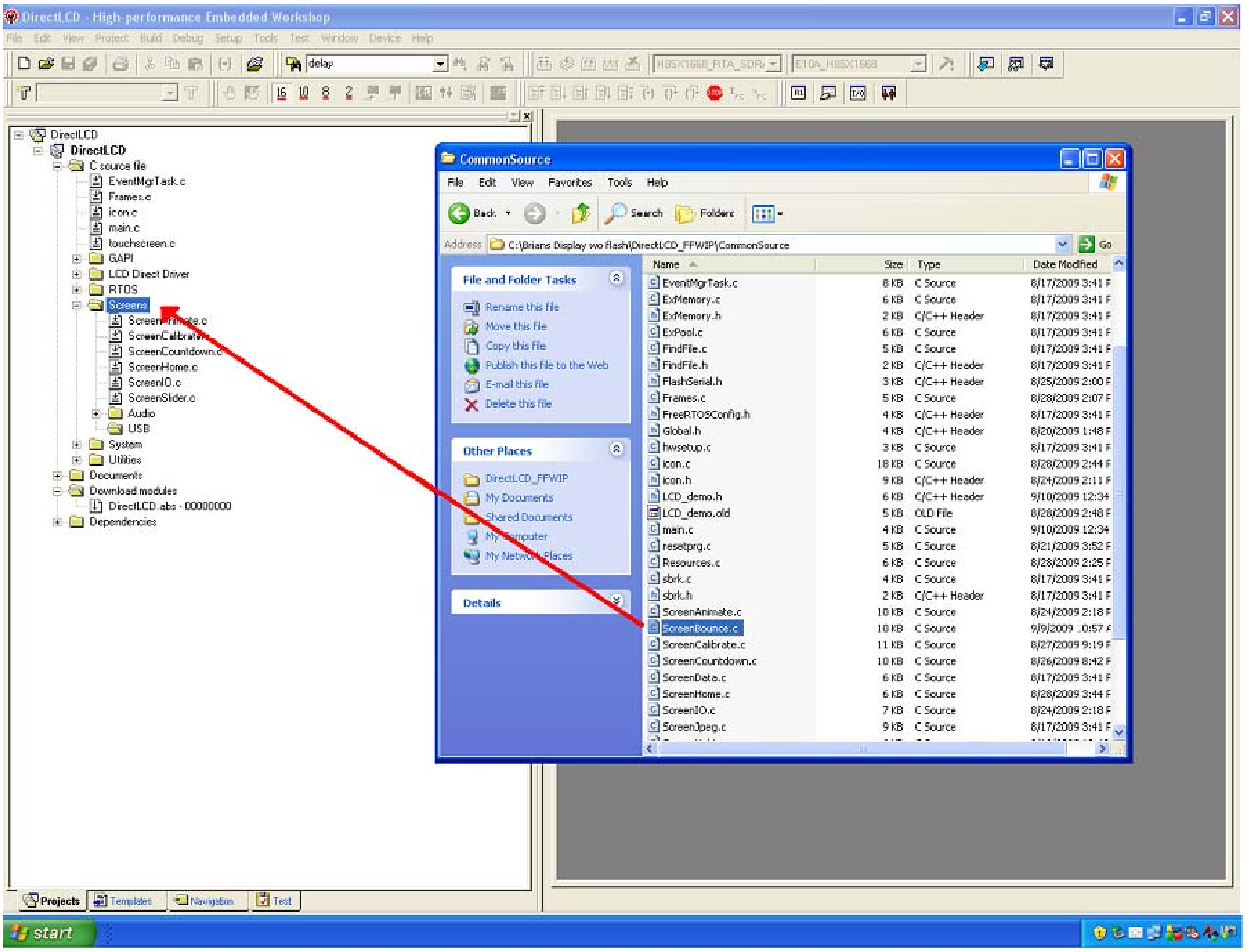


Figure 3: Adding “ScreenBounce.c” to the Direct LCD demo project in HEW

Then, open your LCD Direct Drive demo project in HEW. The ScreenBounce.c file will be located in your CommonSource directory. Add the ScreenBounce.c source file to your project in HEW by dragging the file into the Screens folder of your project. (Figure 3)

The installer places Bitmap Images that are sized for a WQVGA display panel into your resource directory by default. If you are using QVGA or VGA you will need to replace the bitmaps in your resource directory with ones of the appropriate resolution for your display panel. You will find several subdirectories in your resource folder that contain bitmaps of different resolutions. Simply copy all the files from the subdirectories corresponding to your panel resolution and paste them over the files in your resource directory.

Refer to the REU05b0112_H8SAP application note on instructions on how to build and update code resources in the target.

File Name	***File Description***
CommonSource\ScreenBounce.c	Demo screen code
Resources\BigR.bmp	Bitmap image

Website and Support

Renesas Technology Website
http://www.renesas.com/

Renesas Technology America LCD Website
http://america.renesas.com/h8lcd

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com (Global Support)
TechSupport.rta@renesas.com (United States / Canada / Mexico only)

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	March.20.09	—	First edition issued
1.10	January.01.10	—	Converted format to add-in code to REU05b0112

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.